Operating Systems CT-353

LAB 2

Syed Ahab Ali DT-22049

1) Implement the First Come First Serve code and paste the output below.

```c
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main()
{
    int bt[20], wt[20], tat[20], i, n;
    float wtavg, tatavg;
    system("cls");
    printf("\nEnter the number of processes-- ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nEnter Burst Time for Process %d -- ", i);
        scanf("%d", &bt[i]);
    }
    wt[0] = wtavg = 0;
    tat[0] = tatavg = bt[0];
    for (i = 1; i < n; i++)
    {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        wtavg = wtavg + wt[i];
        tatavg = tatavg + tat[i];
    }
    printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
    for (i = 0; i < n; i++)
        printf("\n\t P %d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]);
    printf("\nAverage Waiting Time-- %f", wtavg / n);
    printf("\nAverage Turnaround Time-- %f", tatavg / n);
    getch();
    return 0;
}
```

Output:

```
C:\Users\baziq\OneDrive\Des

Enter the number of processes-- 3

Enter Burst Time for Process 0 -- 5

Enter Burst Time for Process 1 -- 6

Enter Burst Time for Process 2 -- 7
         PROCESS        BURST TIME       WAITING TIME    TURNAROUND TIME

         P 0             5               0               5
         P 1             6               5               11
         P 2             7               11              18
Average Waiting Time-- 5.333333
Average Turnaround Time-- 11.333333
```

2) Implement the Shortest Job First code and paste the output below.

```c
#include <stdio.h>
#include <conio.h>

int main() {
    int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
    float wtavg, tatavg;

    printf("\nEnter the number of processes -- ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        p[i] = i;
        printf("Enter Burst Time for Process %d -- ", i);
        scanf("%d", &bt[i]);
    }
    // Sorting processes based on burst time
    for (i = 0; i < n; i++) {
        for (k = i + 1; k < n; k++) {
            if (bt[i] > bt[k]) {
                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;

                temp = p[i];
                p[i] = p[k];
                p[k] = temp;
            }
        }
    }

    wt[0] = wtavg = 0;
    tat[0] = tatavg = bt[0];

    // Calculating waiting time and turnaround time
    for (i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        wtavg += wt[i];
        tatavg += tat[i];
    }

    printf("\n\tPROCESS\t\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
    for (i = 0; i < n; i++) {
        printf("\n\tP%d\t\t%d\t\t%d\t\t%d", p[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time -- %f", wtavg / n);
    printf("\nAverage Turnaround Time -- %f", tatavg / n);

    getch();
    return 0;
}
```

Output:

```
C:\Users\baziq\OneDrive\Des

Enter the number of processes -- 3
Enter Burst Time for Process 0 -- 3
Enter Burst Time for Process 1 -- 4
Enter Burst Time for Process 2 -- 5

        PROCESS         BURST TIME      WAITING TIME    TURNAROUND TIME
        P0              3               0               3
        P1              4               3               7
        P2              5               7               12
Average Waiting Time -- 3.333333
Average Turnaround Time -- 7.333333
```

3) Implement the Round Robin code and paste the output below.

```c
#include <stdio.h>
#include <conio.h>

int main() {
    int i, j, n, bu[10], wa[10], tat[10], t, ct[10], max;
    float awt = 0, att = 0, temp = 0;

    printf("Enter the no of processes -- ");
    scanf("%d", &n);

    for (i = 0; i < n; i++) {
        printf("\nEnter Burst Time for process %d -- ", i + 1);
        scanf("%d", &bu[i]);
        ct[i] = bu[i];
    }

    printf("\nEnter time quantum -- ");
    scanf("%d", &t);

    max = bu[0];
    for (i = 1; i < n; i++) {
        if (max < bu[i]) {
            max = bu[i];
        }
    }

    for (j = 0; j < (max / t) + 1; j++) {
        for (i = 0; i < n; i++) {
            if (bu[i] != 0) {
                if (bu[i] <= t) {
                    tat[i] = temp + bu[i];
                    temp = temp + bu[i];
                    bu[i] = 0;
                } else {
                    bu[i] = bu[i] - t;
                    temp = temp + t;
                }
            }
        }
    }

    for (i = 0; i < n; i++) {
        wa[i] = tat[i] - ct[i];
        att += tat[i];
        awt += wa[i];
    }

    printf("\nThe Average Turnaround Time is -- %f", att / n);
    printf("\nThe Average Waiting Time is -- %f", awt / n);

    printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
    for (i = 0; i < n; i++) {
        printf("\t%d \t %d \t\t %d \t\t %d \n", i + 1, ct[i], wa[i], tat[i]);
    }

    getch();
    return 0;
}
```

Output:

```
C:\Users\baziq\OneDrive\Des

Enter the no of processes -- 3

Enter Burst Time for process 1 -- 7

Enter Burst Time for process 2 -- 8

Enter Burst Time for process 3 -- 9

Enter time quantum -- 20

The Average Turnaround Time is -- 15.333333
The Average Waiting Time is -- 7.333333
        PROCESS  BURST TIME      WAITING TIME   TURNAROUND TIME
        1        7               0               7
        2        8               7               15
        3        9               15              24
```

4) Implement the Priority Based Scheduling code and paste the output below.

```c
#include <stdio.h>
#include <conio.h>

int main() {
    int p[20], bt[20], pri[20], wt[20], tat[20], i, k, n, temp;
    float wtavg, tatavg;

    printf("Enter the number of processes --- ");
    scanf("%d", &n);
    for (i = 0; i < n; i++) {
        p[i] = i;
        printf("Enter the Burst Time & Priority of Process %d --- ", i);
        scanf("%d %d", &bt[i], &pri[i]);
    }
    for (i = 0; i < n; i++) {
        for (k = i + 1; k < n; k++) {
            if (pri[i] > pri[k]) {
                temp = p[i];
                p[i] = p[k];
                p[k] = temp;

                temp = bt[i];
                bt[i] = bt[k];
                bt[k] = temp;

                temp = pri[i];
                pri[i] = pri[k];
                pri[k] = temp;
            }
        }
    }
    wtavg = wt[0] = 0;
    tatavg = tat[0] = bt[0];

    for (i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
        tat[i] = tat[i - 1] + bt[i];
        wtavg += wt[i];
        tatavg += tat[i];
    }

    printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND TIME");
    for (i = 0; i < n; i++) {
        printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ", p[i], pri[i], bt[i], wt[i], tat[i]);
    }

    printf("\nAverage Waiting Time is --- %f", wtavg / n);
    printf("\nAverage Turnaround Time is --- %f", tatavg / n);

    getch();
    return 0;
}
```

Output:

```
C:\Users\baziq\OneDrive\Des

Enter the number of processes --- 4
Enter the Burst Time & Priority of Process 0 --- 1
2
Enter the Burst Time & Priority of Process 1 --- 3
4
Enter the Burst Time & Priority of Process 2 --- 5
2
Enter the Burst Time & Priority of Process 3 --- 4
2

PROCESS          PRIORITY         BURST TIME       WAITING TIME     TURNAROUND TIME
0                 2                1                0                1
2                 2                5                1                6
3                 2                4                6                10
1                 4                3                10               13
Average Waiting Time is --- 4.250000
Average Turnaround Time is --- 7.500000
```

5) Execute all scheduling algorithms on following data and find out the Average Waiting Time and

Average Turnaround Time of all scheduling algorithms and discuss your results.

(Quantum Value is 3)

Process Name	Burst Time	Priority
P0	2	3
P1	6	1
P2	4	2

FCFS CPU SCHEDULING ALGORITHM

```
C:\Users\admin\Downloads\105.exe
FCFS Scheduling
Process Burst Time      Waiting Time    Turnaround Time
P0      2               0               2
P1      6               2               8
P2      4               8               12
Average Waiting Time: 3.33333
Average Turnaround Time: 7.33333

--------------------------------
Process exited after 0.09211 seconds with return value 0
Press any key to continue . . .
```

SJF CPU SCHEDULING ALGORITHM

```
C:\Users\admin\Downloads\105.exe
SJF Scheduling
Process Burst Time      Waiting Time    Turnaround Time
P0      2               0               2
P2      4               2               6
P1      6               6               12
Average Waiting Time: 2.66667
Average Turnaround Time: 6.66667

--------------------------------
Process exited after 1.969 seconds with return value 0
Press any key to continue . . .
```

PRIORITY CPU SCHEDULING ALGORITHM

```
C:\Users\admin\Downloads\105.exe
Priority Scheduling
Process Burst Time      Priority        Waiting Time    Turnaround Time
P1      6               1               0               6
P2      4               2               6               10
P0      2               3               10              12
Average Waiting Time: 5.33333
Average Turnaround Time: 9.33333


--------------------------------
Process exited after 2.005 seconds with return value 0
Press any key to continue . . .
```

ROUND ROBIN CPU SCHEDULING ALGORITHM

```
C:\Users\admin\Downloads\105.exe
Round Robin Scheduling
Process Burst Time      Waiting Time    Turnaround Time
P0      2               0               2
P1      6               5               11
P2      4               8               12
Average Waiting Time: 4.33333
Average Turnaround Time: 8.33333


--------------------------------
Process exited after 2.121 seconds with return value 0
Press any key to continue . . .
```

Conclusion;

Among the scheduling algorithms, **Shortest Job Next (SJN)** provides the best performance with the lowest Average Waiting Time (AWT) and Average Turnaround Time (ATAT), making it ideal for systems prioritizing quick task completion. **First-Come-First-Serve (FCFS)** is straightforward but can lead to longer waiting times for larger tasks. **Priority Scheduling** is effective for prioritizing critical processes but may increase waiting time for lower-priority tasks. **Round Robin (RR)** ensures fairness and responsiveness in time-shared systems but has slightly higher overhead due to frequent context switching. The choice of the algorithm depends on the system's specific requirements for fairness, efficiency, and priority handling.